Лабораторная работа № 7 по курсу дискретного анализа: Динамическое программирование

Выполнил студент группы М8О-308б-22 МАИ *Караев Тариел Жоомартбекович*.

Условие

Задана матрица натуральных чисел A размерности n × m. Из текущей клетки можно перейти в любую из 3-х соседних, стоящих в строке с номером на единицу больше, при этом за каждый проход через клетку (i, j) взымается штраф Ai,j. Необходимо пройти из какой-нибудь клетки верхней строки до любой клетки нижней, набрав при проходе по клеткам минимальный штраф.

Метод решения

Решение задачи основано на динамическом программировании. Изначально создаются два массива: `dp` для хранения минимальных штрафов и `parent` для восстановления пути. В первой строке матрицы `dp[0][j]` инициализируется значениями штрафов, так как путь начинается с любой клетки верхней строки.

Далее для каждой клетки (i, j) рассматриваются возможные переходы из предыдущей строки: из $(i-1, j-1)$, $(i-1, j)$ и $(i-1, j+1)$, если они существуют. Выбирается переход с наименьшим суммарным штрафом, и массивы `dp` и `parent` обновляются.

После заполнения `dp` находится клетка в нижней строке с минимальным штрафом, и, используя `parent`, восстанавливается путь от нижней до верхней строки. Алгоритм работает за $O(n \cdot m)$.

Описание программы

Код программы:

```
#include <iostream>
#include <vector>
#include <climits>

using namespace std;

int main() {
    long long n, m;
    cin >> n >> m;

    vector<vector<long long>> A(n, vector<long long>(m));
    for (long long i = 0; i < n; ++i) {
        for (long long j = 0; j < m; ++j) {
```

```cpp
            cin >> A[i][j];
        }
    }

    vector<vector<long long>> dp(n, vector<long long>(m, LLONG_MAX));

    vector<vector<long long>> parent(n, vector<long long>(m, -1));

    for (long long j = 0; j < m; ++j) {
        dp[0][j] = A[0][j];
    }

    for (long long i = 1; i < n; ++i) {
        for (long long j = 0; j < m; ++j) {

            for (long long dj = -1; dj <= 1; ++dj) {
                long long prev_j = j + dj;
                if (prev_j >= 0 && prev_j < m) {
                    long long cost = dp[i - 1][prev_j] + A[i][j];
                    if (cost < dp[i][j]) {
                        dp[i][j] = cost;
                        parent[i][j] = prev_j;
                    }
                }
            }
        }
    }

    long long min_cost = LLONG_MAX;
    long long min_index = -1;
    for (long long j = 0; j < m; ++j) {
        if (dp[n - 1][j] < min_cost) {
            min_cost = dp[n - 1][j];
            min_index = j;
        }
    }

    cout << min_cost << endl;

    vector<pair<long long, long long>> path;
    long long i = n - 1;
    while (i >= 0) {
```

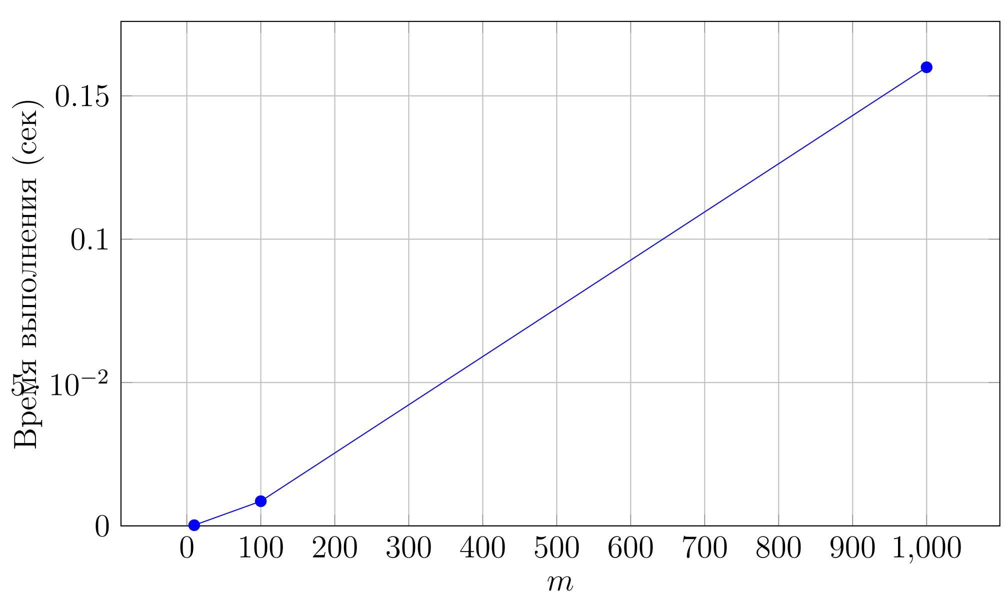
```
            path.push_back({i + 1, min_index + 1});
            min_index = parent[i][min_index];
            --i;
        }


        for (long long i = path.size() - 1; i >= 0; --i) {
            cout << "(" << path[i].first << "," << path[i].second << ") ";
        }
        cout << endl;

        return 0;
    }
```

Дневник отладки

В основном возникали проблемы с реализацией алгоритмов и получением неверных ответов. Один раз был неверно выбран компилятор.

Тест производительности

n	m	Время выполнения (сек)
30	10	0.0002333
300	100	0.0085996
3000	1000	0.159967



Из представленных данных видно, что время выполнения программы растёт линейно с увеличением размеров входных данных, что свидетельствует о линейной сложности алгоритма.

Выводы

В ходе лабораторной работы я реализовал алгоритм динамического программирования для поиска пути с минимальным штрафом в матрице натуральных чисел. Я изучил методы и принципы оптимизации, позволяющие эффективно рассчитывать минимальные суммы штрафов и восстанавливать оптимальный маршрут. Также я провёл замер времени выполнения с помощью библиотеки `chrono` и убедился в линейной зависимости времени работы алгоритма от размеров входных данных. Эта работа углубила мои знания в области динамического программирования и оптимизации алгоритмов.